Schema and Ontology Matching with COMA++

David Aumueller, Hong-Hai Do, Sabine Massmann, Erhard Rahm
Department of Computer Science, University of Leipzig
Augustusplatz 10/11, Leipzig 04103, Germany
{aumueller, hong, massmann, rahm}@informatik.uni-leipzig.de

ABSTRACT
We demonstrate the schema and ontology matching tool COMA++. It extends our previous prototype COMA utilizing a composite approach to combine different match algorithms [3]. COMA++ implements significant improvements and offers a comprehensive infrastructure to solve large real-world match problems. It comes with a graphical interface enabling a variety of user interactions. Using a generic data representation, COMA++ uniformly supports schemas and ontologies, e.g. the powerful standard languages W3C XML Schema and OWL. COMA++ includes new approaches for ontology matching, in particular the utilization of shared taxonomies. Furthermore, different match strategies can be applied including various forms of reusing previously determined match results and a so-called fragment-based match approach which decomposes a large match problem into smaller problems. Finally, COMA++ cannot only be used to solve match problems but also to comparatively evaluate the effectiveness of different match algorithms and strategies.

1. INTRODUCTION
Schema and ontology matching aim at identifying semantic correspondences between metadata structures or *models* such as database schemas, XML message formats, and ontologies. Solving such match problems are of key importance to service interoperability and data integration in numerous application domains. To reduce the manual effort required, many techniques and prototypes have been developed to semi-automatically solve the match problem [6], [11]. The proposed approaches typically exploit metadata (e.g. schema characteristics such as element names, data types and structural properties), characteristics of data instances, as well as background knowledge from dictionaries and thesauri.

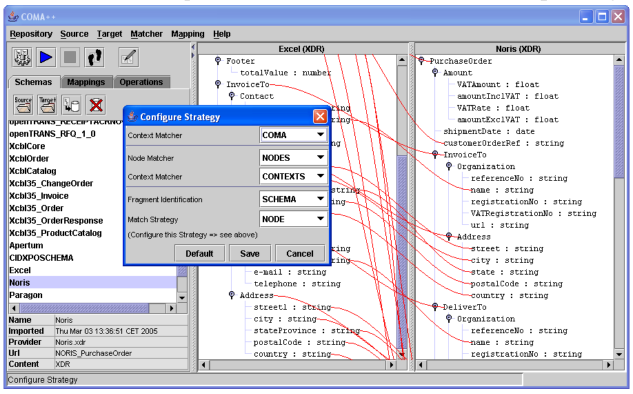
Most prototypes developed so far focus on some research aspects and offer only a rudimentary user interface if any. Given the fact that no fully automatic solution is possible, a user-friendly interface is essential for the practicability and effectiveness of a match system. Furthermore, previous prototypes focus on matching complete input models, which is feasible primarily for small match problems. Although the reuse of previously determined match results promises a significant reduction in manual match work [3], its potential has not yet been fully exploited in current approaches and systems.

COMA++ extends our previous COMA prototype [3] and represents a customizable generic matching tool for both schemas and ontologies. It takes over the flexible composite match approach of COMA to combine different match algorithms, but extends its predecessor with major improvements, namely: comprehensive *graphical user interface*, *generic data model* to uniformly support schemas and ontologies written in different languages, including SQL, W3C XSD and OWL standards, *repository of schemas, ontologies and match results (mappings)* as well as a variety of *high-level operators* on these constructs, e.g. to compose, merge or compare different mappings, a *fragment-based match approach* to decompose a large match problem into smaller problems [10], and *new matchers*, especially for ontology matching and reusing existing match results. Finally, COMA++ can be used as a platform to evaluate different match algorithms. In a comprehensive evaluation, we achieved high quality even on large real-world schemas and ontologies. Due to the highly optimized implementation of the matchers, COMA++ has shown much faster execution times than COMA, especially in large match problems.

The next section provides an overview of the COMA++ architecture and match processing. Section 3 outlines ontology support and Section 4 discusses match strategies in COMA++. We conclude with a brief outlook on what will be demonstrated.

2. OVERVIEW OF COMA++
Figure 1 shows the graphical user interface (GUI) and Figure 2 the underlying architecture of COMA++. The GUI provides access to the five main parts of COMA++, the *Repository* to persistently store all match-related data, the *Model* and *Mapping Pools* to manage schemas, ontologies and mappings in memory, the *Match Customizer* to configure matchers and match strategies, and the *Execution Engine* to perform match operations.

To maximize the potential for reuse [7], [3], the **Repository**



Context Matcher	COMA
Node Matcher	NODES
Context Matcher	CONTEXTS
Fragment Identification	SCHEMA
Match Strategy	NODE

(Configure this Strategy => see above)

openTRANS_RFQ_1_0
XcblCore
XcblOrder
XcblCatalog
Xcbl35_ChangeOrder
Xcbl35_Invoice
Xcbl35_Order
Xcbl35_OrderResponse
Xcbl35_ProductCatalog
Apertum
CIDXPOSCHEMA
Excel
Noris
Paragon

Name	Noris
Imported	Thu Mar 03 13:36:51 CET 2005
Provider	Noris.xdr
Url	NORIS_PurchaseOrder
Content	XDR

```
Footer
  totalValue : number
InvoiceTo
  Contact
    e-mail : string
    telephone : string
  Address
    street1 : string
    city : string
    stateProvince : string
    postalCode : string
    country : string
```

```
PurchaseOrder
  Amount
    VATAmount : float
    amountInclVAT : float
    VATRate : float
    amountExclVAT : float
  shipmentDate : date
  customerOrderRef : string
  InvoiceTo
    Organization
      referenceNo : string
      name : string
      registrationNo : string
      VATRegistrationNo : string
      url : string
    Address
      street : string
      city : string
      state : string
      postalCode : string
      country : string
  DeliverTo
    Organization
      referenceNo : string
      name : string
      registrationNo : string
```

Figure 1. User interface of COMA++

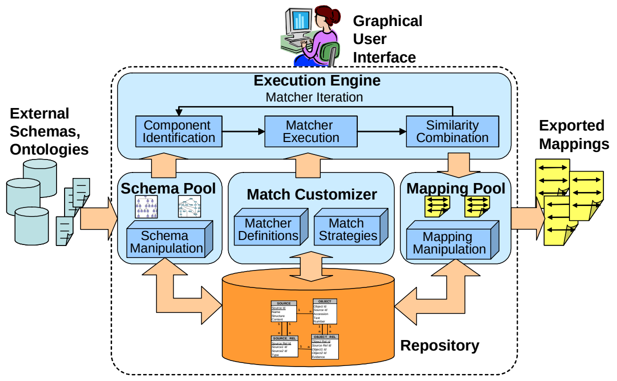


Figure 2. Architecture of COMA++

centrally stores various types of data related to match processing, in particular imported schemas and ontologies, produced mappings, auxiliary information such as domain-specific taxonomies and synonym tables, and the definition and configuration of the matchers. We use a generic data model implemented in a relational DBMS (currently MySQL) to uniformly store the different kinds of schemas and ontologies as well as mappings between them.

Models are uniformly represented by directed graphs as the internal format for matching. The **Model Pool** provides different functions to import external schemas and ontologies, and to load and save them from/to the repository. Formats supported by COMA++ include XSD, XML Data Reduced (XDR), OWL, and relational schemas. From the Model Pool, two arbitrary models can be selected to start a match operation.

Automatic match processing is performed in the **Execution Engine** in the form of *match iterations*, which are the building blocks for match strategies such as fragment-based matching. As indicated in Figure 2, match iterations take place in three steps, *component identification* to determine the relevant schema components for matching, *matcher execution* applying multiple matchers to compute component similarities, and *similarity combination* to combine matcher-specific similarities and derive the correspondences between the components. The obtained mapping can be used as input in the next iteration for further refinement. Each iteration can be individually configured using the alternatives supported by the **Match Customizer**, i.e. the types of components to be considered, the matchers for similarity computation, and the strategies for similarity combination.

COMA++ supports various methods to determine the components of a schema, such as nodes, paths, and fragments, as well as to determine the constituents of a single component, such as its name tokens, child nodes, etc., which can be considered to estimate the similarity between two components. Multiple matchers can be selected from the *Matcher Library* to compute the similarity between the identified components, resulting in a similarity cube. Currently, more than 15 matchers exploiting different kinds of schema and auxiliary information are available. COMA++ then employs the combination scheme developed in COMA with corresponding strategies for the sub-steps *aggregation*, *direction*, and *selection* [3] to derive a match result from the similarity cube. The obtained mapping is a set of correspondences specifying the matching components between two input models. Each pair of matching components is captured in a single correspondence, i.e. a 1:1 match. Since components may occur in multiple correspondences, matches with n:m cardinality are possible, too.

The **Mapping Pool** maintains all generated mappings and offers various functions to further manipulate them, such as to determine the different/common correspondences between two mappings (*Diff*/*Intersect*), to merge two mappings (*Merge*), to transitively combine mappings sharing a same schema (*MatchCompose* [3]), and to evaluate the decency of a test mapping against an expected mapping according to different quality measures (*Compare*). Users can also edit each mapping to remove false correspondences or to add missing ones by clicking on the schema components shown within the GUI (*Edit*). The Mapping Pool offers further functions to load and save mappings from/to the repository, as well as to import and export them using an XML/RDF and a CSV format.

3. ONTOLOGY SUPPORT

OWL Support. COMA++ currently supports matching between ontologies written in W3C OWL-Lite. OWL class hierarchies and relationship types are read in via the OWL API [1] and mapped to the generic model representation based on directed acyclic graphs. While OWL/RDF (Resource Description Framework) nodes are identified via URIs, as node names in COMA++ we use the RDF label if available, otherwise the last URI fragment. After importing the OWL ontologies we can visualize them like schemas and apply all available matchers, in particular various name and structural matchers. We tested COMA++ on a set of recently proposed ontology match problems (http://co4.inrialpes.fr/align/Contest/). Even without providing domain-specific taxonomies or synonyms, COMA++ solved most problems with high accuracy.

Taxonomy Matcher. The new taxonomy matcher matches two schemas or ontologies with the help of a taxonomy that gets linked between the source and target of the match task. The taxonomy thus acts as an intermediary ontology, which is used as a reference to look up the relatedness of source and target components – similarly as proposed in [8]. We believe the use of domain-specific taxonomies is much more promising than applying huge general dictionaries such as WordNet which likely cause poor precision in many cases.

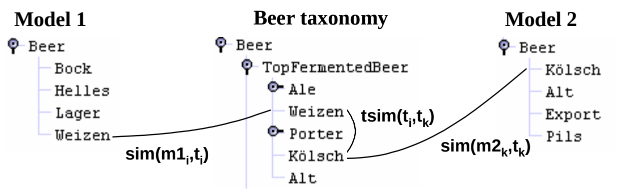


Figure 3. Taxonomy-based matching

The similarity measure between two models is determined by a function of the similarity (distance) between the two matching elements *within* the taxonomy, $tsim(t_i,t_k)$, combined with the match similarity of the two matching models with the taxonomy. In the example of Figure 3, the elements *Weizen* and *Kölsch* of Model 1 and 2, respectively, are hyponyms of *top fermented beer* in the given *Beer taxonomy*. That is, they share the same hypernym and the matcher assigns a similarity value dependent on the distance between the two terms within the taxonomy. To determine the intra-taxonomy similarity *tsim* we are still experimenting with various alternatives, e.g. those of [2] and [12].

4. MATCH STRATEGIES

COMA++ supports higher-level strategies to address complex match problems, in particular fragment-based matching and the reuse of previous match results.

Fragment-based Matching. To cope with large schemas, COMA++ implements the fragment-based match processing framework proposed in [10]. Following the divide-and-conquer idea, it decomposes a large match problem into smaller sub-problems by matching at the level of schema fragments. With the reduced problem size, we aim not only at better execution time but also at better match quality compared to schema-level matching. Our framework encompasses two matching phases:

1. *Identify similar fragments.* Depending on a specified fragment type, this step determines the fragments from the input schemas and compares them to identify the most similar ones worth to be fully matched later. Supported fragment granularities include complete models, so-called subschemas as independent parts of a schema (e.g. XML message formats), shared structures, and user-specified fragments.
2. *Match fragments.* Each pair of similar fragments represents an individual match problem, which is solved in a single match operation to identify correspondences between their components. The result is a set of mappings containing correspondences between fragment components, which are then merged into a global match result.

In the automatic mode, two phases are executed in a single pass using pre-specified strategies. COMA++ also supports step-by-step fragment matching, allowing the user to verify and make changes to the outcome of each phase.

Reuse-oriented Matching. The reuse of existing schemas is addressed in [7], focusing on learning and using component statistics in a corpus of schemas for matching. In contrast, we pursue the reuse of previously determined match results. The main mechanism for our approach is the MatchCompose operation [3], which performs a join-like operation on a mapping path consisting of two or more mappings, such as A-B, B-C, and C-D, successively sharing a common schema, to derive a new mapping between A and D. While COMA presupposes the existence of such mapping paths, COMA++ now also considers cases where such mapping paths are unnecessary (i) or not available (iii):

i. *Direct mappings.* This is the ideal case for reuse, in which one or multiple mappings are already available for the given match problem.

ii. *Complete mapping paths.* In this case, COMA++ searches in the repository for all mapping paths of different lengths, 2, 3, etc. (i.e. the number of involved mappings), connecting the two input schemas with each other.

iii. *Incomplete mapping paths.* In this case, COMA++ searches for mapping paths involving mappings which are not available yet, but may be computed with less effort than directly matching the input schemas. For instance, if A was already matched to B', an older version of B, we can combine this existing mapping with the match result for B'-B to solve our task A-B. This assumes it is easier to newly match B'-B than A-B, because only few B components are expected to change compared to B'.

The last strategy increases the potential for reuse, but also requires some heuristics to determine such 'light-weight' match tasks. For this purpose, we search for existing mappings involving models which are similar to at least one of the input models. This search for similar models is a variation of the identification of similar fragments in fragment-based matching. Currently, we consider metadata such as the schema version, used namespaces, and some simple string matching algorithms on the model names.

COMA++ further supports the utilization of a so-called *pivot schema*, e.g. a standard schema or ontology in a domain, acting as the central schema, against which all new schemas are first matched. Matching any two schemas can be efficiently performed by reusing the mappings between them and the pivot schema, resulting in mapping paths with maximal length of 2.

From the proposed reuse possibilities, the user may choose one or combine multiple ones to derive a match result. In the automatic mode, COMA++ ranks the identified possibilities according to the expected effort by looking at the path length and the size of the involved mappings and uses a predefined number of best mapping paths. The mappings derived from the mapping paths represent possible results for the given match task. Similar to the match results obtained using multiple matchers, they constitute a similarity cube and can also be combined to complement each other by means of the combination scheme (see Section 2).

5. TOOL DEMONSTRATION

We will apply different matchers and match strategies in COMA++ to match between schemas and ontologies from several domains. Thereby, various interaction possibilities to influence the match process will be demonstrated, such as configuration of matchers and match strategies, step-by-step execution of match operations, verification and further manipulation of match results.

6. ACKNOWLEDGMENTS

Hong-Hai Do is supported by DFG grant BIZ 6/1-1.

7. REFERENCES

[1] Bechhofer S., R. Volz, P. Lord: *Cooking the Semantic Web with the OWL API.* International Semantic Web Conference (ISWC) 2003

[2] Budanitsky A.: *Lexical Semantic Relatedness and Its Application in Natural Language Processing.* Tech. Report, Univ. Toronto, 1999

[3] Do, H.H., E. Rahm: *COMA – A System for Flexible Combination of Match Algorithms.* VLDB 2002

[4] Do, H.H., S. Melnik, E. Rahm: *Comparison of Schema Matching Evaluations.* Proc. Workshop Web and Databases, LNCS 2593, 2003

[5] Doan A., J. Madhavan, P. Domingo, A. Halevy: *Learning to Map between Ontologies on the Semantic Web.* WWW 2002

[6] Kalfoglou, Y., M. Schorlemmer: *Ontology Mapping - The State of The Art.* Knowledge Engineering Review 18(1), 2003

[7] Madhavan, J., P.A. Bernstein, A.H. Doan, A.Y. Halevy: *Corpus-based Schema Matching.* Int. Conf. of Data Engineering (ICDE) 2005

[8] Mena, E. et al: *Managing Multiple Information Sources through Ontologies: Relationship between Vocabulary Heterogeneity and Loss of Information.* Knowledge Representation Meets Databases (KRDB) 1996

[9] Popa, L., M. Hernández, Y. Velegrakis, R. Miller: *Mapping XML and Relational Schemas with Clio.* ICDE 2002 (Demonstration)

[10] Rahm, E., H.H. Do, S. Massmann: *Matching Large XML Schemas.* SIGMOD Record 33(4), 2004

[11] Rahm, E., P.A. Bernstein: *A Survey of Approaches to Automatic Schema Matching.* VLDB Journal 10 (4), 2001

[12] Weeds, J., D. Weir, D. McCarthy: *Characterising Measures of Lexical Distributional Similarity.* Int. Conf. of Computational Linguistics (COLING) 2004